A Comparison of Library Tracking Methods in High Performance Computing

Los Alamos NATIONAL LABORATORY

Dennis Trujillo, Chris DeJager, William Rosenberger

Mentors: Georgia Pedicini (HPC-3) and **David Gunter** (HPC-3) **Instructor:** Dane Gardner (NMC)

PRObE Parallel Reconfigurable Observational Environment

Introduction

High Performance Computing (HPC) software support teams must maintain multiple versions of the same libraries to ensure code compatibility and consistent results on long running jobs. Maintaining unused libraries is a waste of resources, but it is difficult to find unused libraries without a tracking system. We selected the Automatic Library Tracking Database (ALTD) and the Linux Auditing Utility (auditd) as potential tracking solutions. ALTD was selected because of its capability, and auditd was selected because of its maturity.

With a capable tool, system administrators would then be able to generate data similar to Figure 1 and Figure 2 providing insight about which libraries can be removed. In Figure 1 openmpi-1.6.5 is introduced to the system and slowly adopted by the user base. The administrators can then find the individuals still using 1.5.4 and encourage them to move to 1.6.5. In Figure 2 some users try to adopt a new version of gcc but decide to move back to the old version, possibly due to a bug.

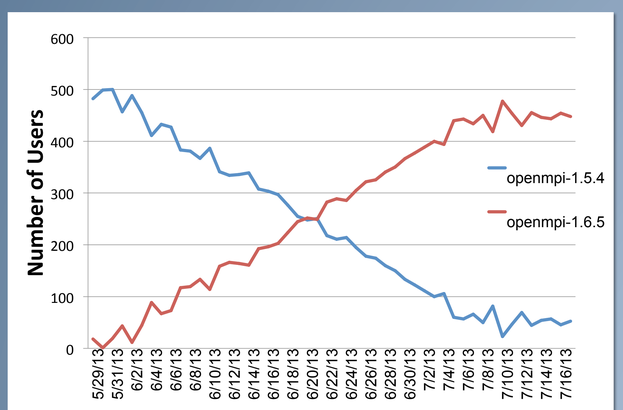

Figure 1: Simulated New Software Version Acceptance

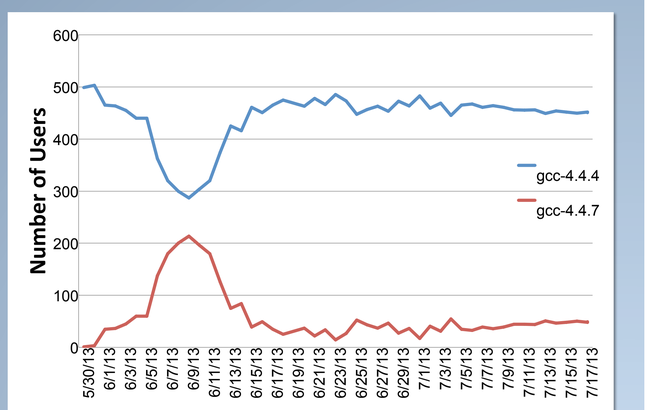

Figure 2: Simulated New Software Version Rejection

Linux Auditing Utility

The Linux Auditing Utility auditd is a Linux Kernel daemon included by default in many Linux distributions. Auditd tracks library usage by watching for when individual files are accessed. System administrators must place an auditd watcher on every library to be tracked. Although originally designed for creating an audit trail describing the use of secured system resources, auditd is capable of following individual library files to watch for when they are used. Conveniently, auditd includes command line tools for interacting with the log file. Using these tools, we are able to search for usage instances for individual libraries.

Automatic Library Tracking Database

The Automatic Library Tracking Database was originally presented at the Cray Users Group with the goal of creating a low-overhead solution for tracking library usage. We expand upon their original product to provide a greater range of functionality and make the system more user friendly.

As illustrated in Figure 3, ALTD makes use of a series of scripts to wrap the job launcher and Linux linker in order to collect data on when libraries are used. Information on which libraries were linked into a program, as well as when that program is run is stored in a MySQL database. We have expanded on the number of supported job launchers, as well as created tools to easily query the database to get a count of each time a library is used.

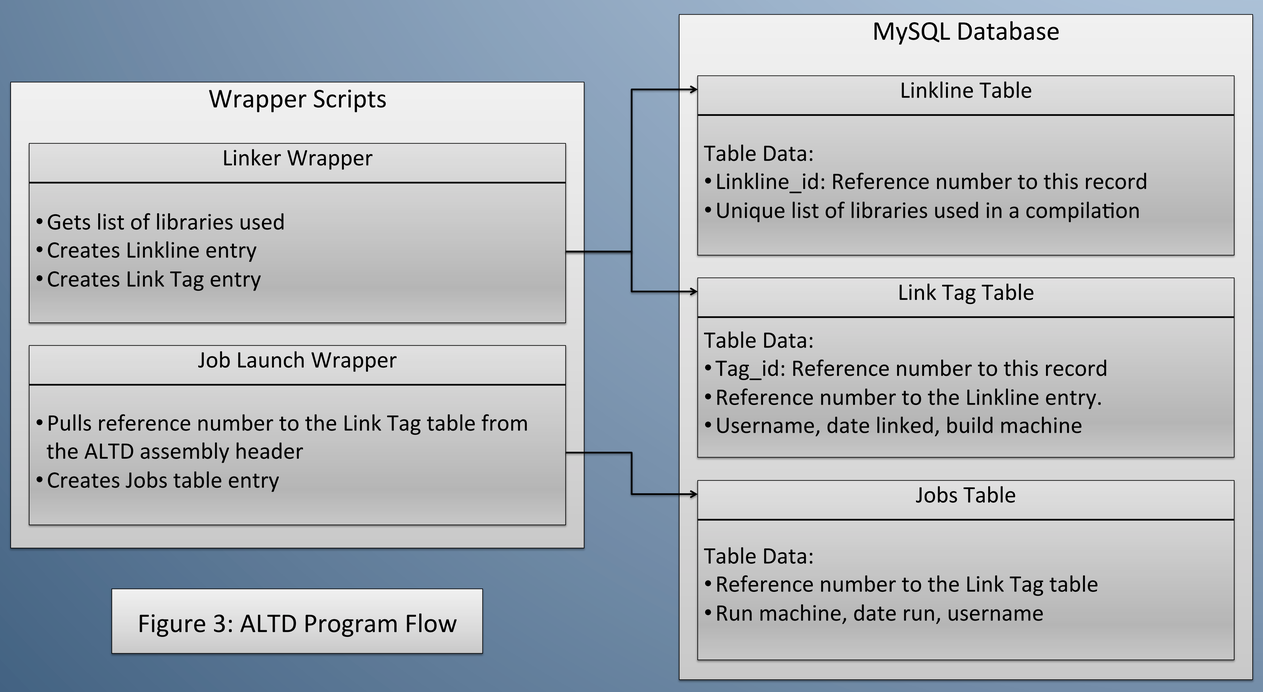

Figure 3: ALTD Program Flow

Performance Testing

The increase in compilation and run time caused by ALTD and auditd was studied to determine system impact. The compilation was timed for a "do nothing" MPI program as well as with the more complicated Linpack in Figure 4. Although auditd outperformed ALTD for the simple program, auditd's compilation overhead increased significantly as the program became more complex. Figure 5 illustrates the increase in runtime for the "do nothing" MPI program. Increase in runtime of Linpack caused by ALTD and audit is illustrated in Figure 6 and 7 respectively. ALTD has an average increase of 0.1209 seconds while auditd has an average increase of 0.0030 seconds. None of our data suggests a relationship between increase in runtime and number of processors problem size. Therefore from a performance standpoint ALTD and auditd are both viable.

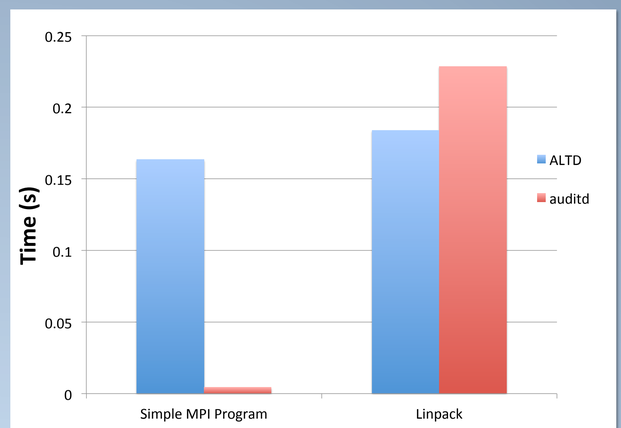

Figure 4: Additional Compilation Time Required

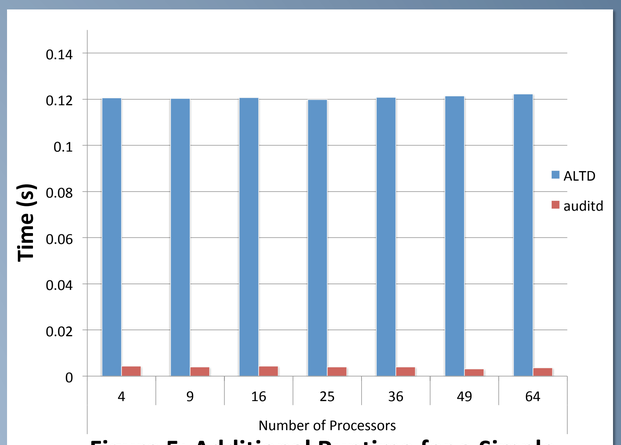

Figure 5: Additional Runtime for a Simple MPI Program

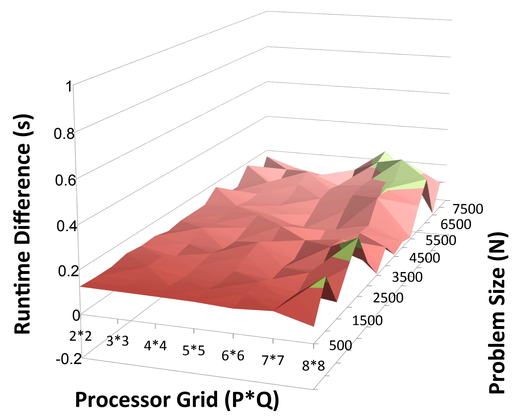

Figure 6: Additional Runtime for Linpack With ALTD

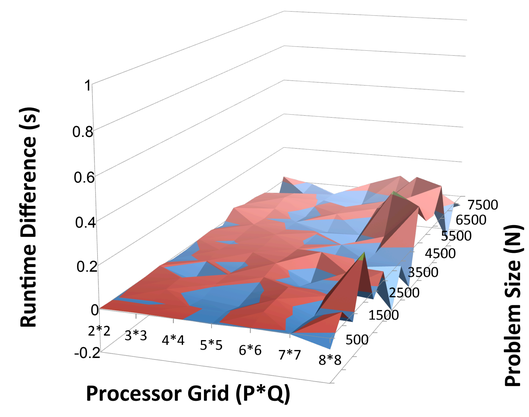

Figure 7: Additional Runtime for Linpack With auditd

ALTD vs. auditd

Automatic Library Tracking Database

Pros
- Data is well organized in a MySQL database
- Theoretical constant low overhead
- Track all libraries automatically

Cons
- Requires a MySQL database
- Track dynamic libraries used at compile time, not necessarily runtime

Linux Auditing Utility

Pros
- Standard Linux Daemon
- Well tested by the Linux community
- Logs to a flat file

Cons
- Dynamically linked libraries can't be reliably tracked, the .so files are cached in memory and not read at every run
- Statically linked libraries can't be track at runtime as they are in the executable
- A single make can generate numerous log entries

Conclusion

We found over the course of testing that the Automatic Library Tracking Database is the best solution out of those investigated. From a performance standpoint both ALTD and auditd are viable but, ALTD tracks more types of libraries and does this automatically. Additions made to the ALTD software have allowed the program to run on a larger variety of systems, and improvements have been made to user interaction with the database. With respect to an HPC setting, many libraries are difficult or impossible to track with auditd preventing it from being a viable tracking option.

Contact Info

Denis Trujillo : dptru10@nmsu.edu
Chris DeJager: cadejage@mtu.edu
William Rosenburger: wrosenberger@live.com

LA-UR-13-25964

ISTI INFORMATION SCIENCE & TECHNOLOGY INSTITUTE
NSF
Michigan Technological University
NEW MEXICO TECH SCIENCE • ENGINEERING • RESEARCH UNIVERSITY
NM STATE UNIVERSITY
New Mexico CONSORTIUM